Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Szakszon Mátyás

GÉPJÁRMŰKÖVETÉS A MESTERSÉGES INTELLIGENCIA MÓDSZEREIVEL

KONZULENS

Alekszejenkó Levente

BUDAPEST, 2015

Tartalomjegyzék

1 Bevezetés

ÖNLABBÓL van még TODO lesz átírni

Manapság egyre nagyobb figyelmet kap a mesterséges intelligencia, ami nem is meglepő, mert rengeteg eddig megoldatlan problémára egyszerű és gyors megoldást adnak. Ezen problémák körébe tartozik a közlekedés fejlesztése, ami kiemelten fontos a mai világban, mert egyre több az olyan nagy város, ahol egész nap áll a forgalom és emiatt az emberek rengeteg időt veszítenek el. Ezen probléma esetében a cél az, hogy a jelenleg rendelkezésre álló utak, táblák, lámpák és a közlekedési szabályok segítségével a legoptimálisabb kombinációt hozzák létre, amivel a legnagyobb mértékben gyorsul a közlekedés. Ezen probléma esetében könnyen belátható, hogy olyan sok változótól függ, a feladat megoldása, hogy ez egy ember számára emberi időn belül megoldhatatlan, azonban egy jól betanított mesterséges intelligencia, az optimumhoz közeli megoldást tud adni, elfogadható időn belül, ami hatalmas előrelépést jelent az ilyen típusú feladatok esetében.

Az én témám is a közlekedésre épül, de egy kevésbé kutatott témára épül. Én egy közlekedésből véletlenszerűen kiválasztott járműnek az útvonalát szeretném meghatározni, minél pontosabban, minél kevesebb információ felhasználásával.

Idén kipróbálom, hogy kizárólag autókhoz tartozó útvonalak állnak rendelkezésemre, akkor milyen eredményt tudok elérni, az előző évhez képest ez annyiban nehezítés, hogy akkor rendelkezésünkre állt, az úthálózat felépítése és az arra vonatkozó statisztikák.

2 Előzetes ismeretek

2.1 Python

PythonHiba! A hivatkozási forrás nem található. egy magas szintű, általános célú programozási nyelv, amelyet könnyen érthető és olvasható szintaxis jellemzi. Előnye, hogy egyszerűen használható és széles körben elterjedt a szoftverfejlesztésben, tudományos kutatásban, adatelemzésben és számos más területen.

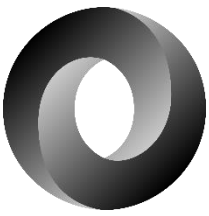
Könnyen tanulható és hatékony programozási nyelv, amely sokoldalúságával és széleskörű könyvtárainak támogatásával rendkívül népszerű a fejlesztők körében. A szintaxisa és a magas absztrakciós szintje révén lehetővé teszi a programozók számára, hogy hatékonyan és produktívan dolgozzanak különböző alkalmazások és projektek fejlesztése során.

2.2 Visual Studio Code (VS Code)

Az Visual Studio Code (VS Code)**Hiba! A hivatkozási forrás nem található.** egy ingyenesen elérhető, nyílt forráskódú fejlesztői környezet, amelyet kifejezetten az alkalmazásfejlesztők számára terveztek. Könnyen testreszabható és bővíthető, és támogatja a számos programozási nyelvet és keretrendszert.

Rendelkezik egy intuitív felhasználói felülettel, amelyet könnyű kezelni. Az editor lehetővé teszi a szövegszerkesztési funkciók, például a kódkiegészítés, a szintaxiskiemelés és a formázás használatát, hogy hatékonyan és hibamentesen írhassunk kódot. Az integrált hibakeresés és hibakereső eszközök segítségével könnyedén megtalálhatjuk és javíthatjuk a hibákat a kódunkban.

2.3 JSON (JavaScript Object Notation)



A JSONHiba! A hivatkozási forrás nem található. egy könnyen olvasható és írható adatállomány formátum, amelyet gyakran használnak adatok strukturált reprezentálására és azok közvetítésére a weben. A JSON adatok emberi olvasható formában vannak, és könnyen feldolgozhatók és generálhatók különböző programozási nyelvekben.

A JSON adatokat objektumok és tömbök hierarchiájában szervezik. Az objektumokat kapcsos zárójelek között adhatjuk meg, ahol kulcs-érték párokat tartalmaznak. A kulcsokat szöveg formájában adjuk meg, és az értékek lehetnek szövegek, számok, logikai értékek, null, más objektumok vagy tömbök. A tömbök négyzetes zárójelek között találhatók, és tartalmazhatnak bármilyen típusú értéket.

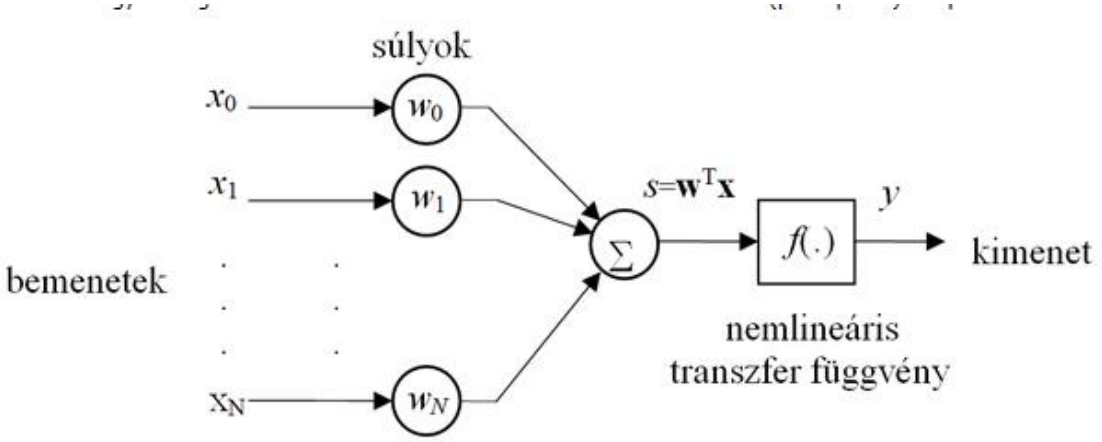
2.4 Neurális hálózat

A neurális hálózat (vagy mesterséges neurális hálózat)**Hiba! A hivatkozási f orrás nem található.** egy olyan számítási modell, amely az emberi agy működéséből inspirálódik. A neurális hálózatokat alkalmazzák gépi tanulási feladatokra, például mintázatfelismerésre, osztályozásra, regresszióra, klaszterezésre és egyéb feladatokra.

Egy neurális hálózat számos egyszerű egységből vagy "neuronból" épül fel, amelyek az emberi idegsejtek (neuronok) működését modellezik. Ezek a neuronok rétegekbe szerveződnek, és kapcsolatokkal vannak összekötve, amelyek súlyokkal vannak ellátva. Az első réteg a bemeneti réteg, az utolsó réteg pedig a kimeneti réteg. A köztes rétegek pedig "rejtett rétegek".

A neurális hálózatokban a tanulás azáltal történik, hogy a rendszer beállítja a kapcsolatok súlyait a bemeneti adatok és a kimeneti válaszok alapján. A tanítás során a hálózat predikciót hoz létre, majd a predikció hibáját visszaterjeszti a hálózatba a hibavisszaterjesztés (backpropagation) algoritmus segítségével. A súlyokat úgy állítják be, hogy minimalizálják a predikció és a valóságos válasz közötti hibát.

A neurális hálózatoknak különböző architektúráik vannak, beleértve az egyszerűbb típusokat, mint a többrétegű perceptronok, és bonyolultabbakat, mint a konvolúciós neurális hálózatok (CNN) vagy a rekurrens neurális hálózatok (RNN). Az ilyen különböző típusú hálózatoknak különböző felhasználási területeik vannak, és alkalmazzák őket a képfelismeréstől és a nyelvi modellezésig.



1. ábra Neurális hálózat
(http://project.mit.bme.hu/mi_almanach/books/neuralis/ch01s02)

3 Feladat

Ebben a félévben az volt a feladatom, hogy a rendelkezésre álló adatot feldolgozzam és egy modellt tanítsak rajta. Az adat autókat és azokhoz tartozó útvonalakat tartalmazott. Mint modell a VAE-ra esett a választás, aminek a célja, hogy a bemenetként érkező útvonalakat egy látens térbe kódolja és onnan minél pontosabban visszaállítsa az bemenetre érkező adatot. Ezért most a cél nem az volt, hogy a beérkező útvonalakat folytassam tovább, hanem hogy az Encoder leválasztásával és a látens tér segítségével új útvonalakat generáljak. Új útvonal alatt olyat utakra gondolunk, amiket a tanító adat nem tartalmaz, azonban az eredeti térkép tartalmaz.

Tehát az erre félévre kitűzött cél egy Variációs Autoencoder betanítása, a bemeneti adat minél pontosabb visszaállítása és új adat generálása.

4 Adat augmentáció

4.1 Adatstruktúra

A rendelkezésre álló adat kulcsérték párokból épült fel, ahol a kulcs egy autónak az ID-ja és a hozzá tartozó érték egy lista, ami utcáknak az ID-ját tartalmazta. A lista sorrendben tartalmaz utcákat, amik együtt egy útvonalat határoz meg, amin az autó közlekedett.

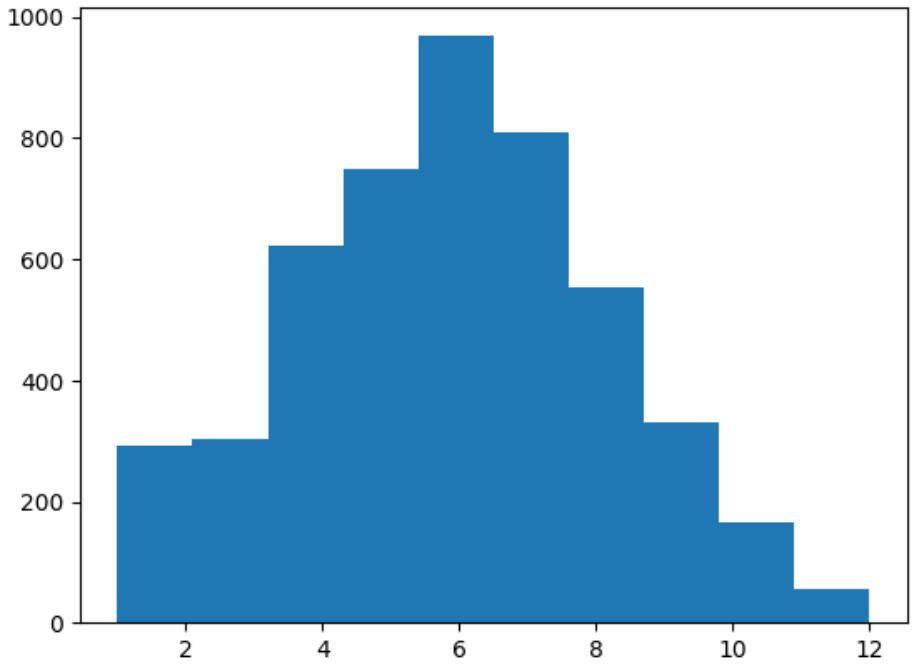
2. ábra Adatstruktúra

A munkám folyamán az adatot többféleképpen augmentáltam. Ezek a változtatások azt a célt szolgálták, hogy megtaláljam az adatnak egy olyan reprezentációját, amit az álltalám választott modellt a legjobban segíti abban, hogy rátanuljon az adatban található összefüggésekre.

4.2 Adat egységesítése

Kezdetből szükségem volt arra, hogy egységesítsem az adatot, ami azt jelenti, hogy minden adatnak ugyanúgy kell kinézni jelen esetben ez azt jelenti, hogy az utakat egységes hosszúságra kell hozni, hogy a modellnek át lehessen adni.

Az utak hosszának eloszlása az alábbi ábráról leolvasható.



3. ábra Utak eloszlása az adathalmazban

A legtöbb a 6 hosszú utakból van és minden út a 2 és 12 hosszú tartományba esik. Ebből következően az egységes hosszúság a 6 lesz, ami azt jelenti, hogy minden 6-nál rövidebb utat elhagyunk az adathalmazból és minden 6-nál hosszabb utat egy csúszó ablak segítségével adat dúsítunk.

A 6-nál rövidebb utakat elhagyva vesztünk valamennyi információt, azonban ez nem okoz problémát, mert a hosszabb utak adat dúsításával összeségében több utunk lesz, mint ami eleve rendelkezésre állt, illetve a hosszabb utak nagy valószínűséggel tartalmazták a rövidebb utakat részhalmazaikként, ezért kevesebb információt vesztünk, mint azt gondolnánk.

Eredetileg **4849** különböző utunk volt, ami az egységesítés végén **6730**-ra bővült.

4.3 One Hot Encoding

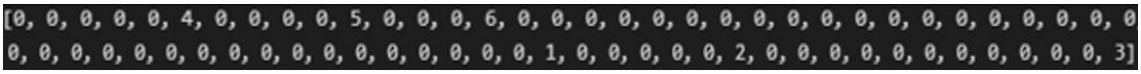
A One Hot Encoding kategorikus változok kódolására alkalmazzák, ami azt a célt szolgálja, hogy modell a kategóriákat egyenértékűnek tekintse. Ez a probléma azért merül fel, mert amikor valamit ID-val reprezentálunk az a modell számára csak egy szám és mi szeretnénk azt az információt átadni a modellnek, hogy nem számít a szám értéke, az csak egy egyedi azonosító, erre szolgál a One Hot Encoding.

4.3.1 Működése

Minden kategóriás változót egységen egy olyan vektorral ábrázolunk, aminek a hossza a különböző kategóriák száma és kizárólag 0-ákat tartalmaz és 1db 1-est azon a helyiértéken, ami azt az adott kategóriát reprezentálja.

4.3.2 One Hot Endcoding másképp

Jelen esetünkben először kísérletezésként kipróbáltam egy egyedi megközelítését a One Hot Encoding-nak, ami úgy működött, hogy a vektor nem csak 0-ákat és 1db 1-est tartalmazott, hanem megpróbáltam a teljes 6 hosszú utamat 1db vektorba lekódolni oly módon, hogy az első utca ID-ja helyén 1-es állt a 2. utca ID-ja helyén 2-es és így tovább egészen 6-ig, minden más maradt 0.



4. ábra One Hot Encoding másképp

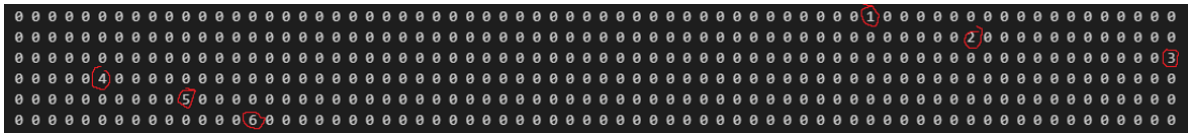
Erre azért volt szükség, mert valahogy szükséges átadnom azt az információt a modell számára, hogy mi az utcák sorrendje, különben nincs milyen összefüggést megtanulnia. Azonban ez ahhoz a problémához vezetett, hogy nem használhattam Sigmoid aktivációs függvényt a Decoderem utolsó rétegéként, hanem az alapján választottam ki a predikált útvonalat, hogy a kiválasztottam sorban a 6 legnagyobb értéket a predikált vektorból.

Azonban az így tanított modellem nagyon alacsony pontossággal rendelkezett, aminek az okát nem tudtam pontosan megállapítani, ezért szerettem volna, a sima One Hot Encoding-hoz visszatérni, ami azt is jelentette, hogy a modellemben az Encoder utolsó aktivációs függvényét Sigmoidra cserélhettem.

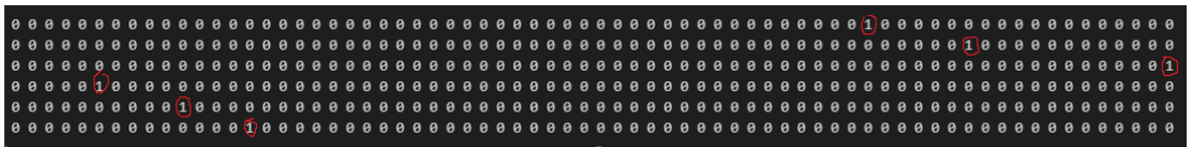
4.3.3 Az Adat One Hot Encodolása

Hogyan tudnám One Hot Encodolni az adatomat úgy, hogy megőrzőm a sorrendiséget? Arra a döntésre jutottam, hogy az egy 70 hosszú vektoromat átalakítom 6db 70 hosszú vektorrá úgy, hogy minden 70 hosszú vektor csak 1db 1-est fog tartalmazni és minden más 0. A sorrendiséget pedig az reprezentálja, hogy egymás után milyen sorrendben szerepel a 6 db vektor.

Ezt az átalakítást 2 lépésben végeztem el, ami az alábbi ábrákon látható.



5. ábra One Hot Encoding 1. lépés



6. ábra One Hot Encoding 2. lépés

Most már az adatom minden feltételét teljesítette a One Hot Encodolásnak. Ez volt az adatom végső formája, a végén elért eredményeket ezzel produkáltam.

5 Megismert modellek/módszerek

5.1 Variációs Autoencoder (VAE)

5.1.1 Mi az a VAE?

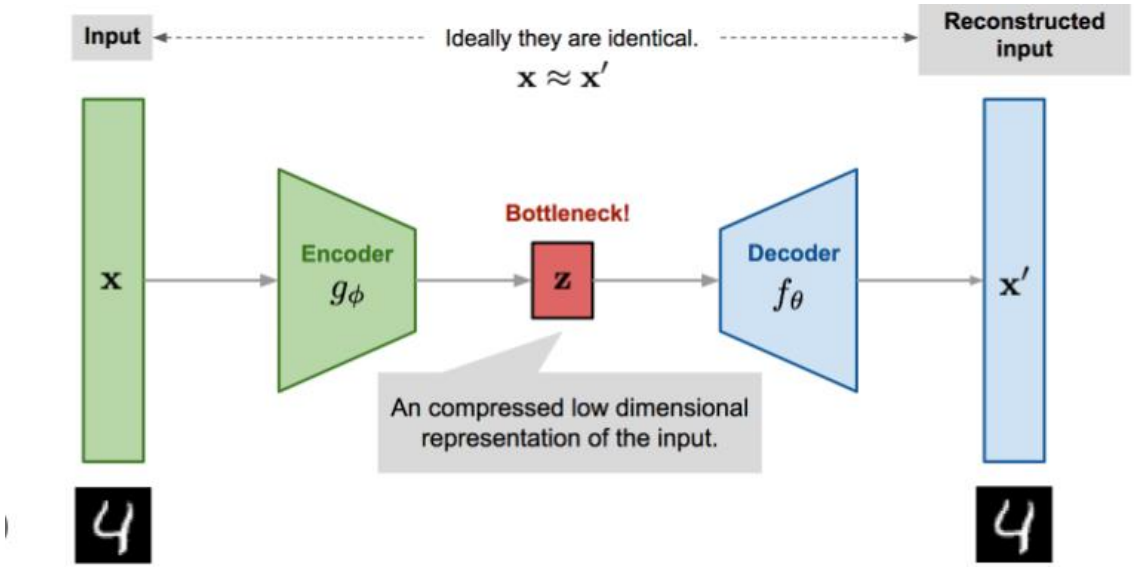
A Variációs Autoencoder (VAE)**Hiba! A hivatkozási forrás nem található.** egy olyan típusú mesterséges neurális hálózat, amely a generatív modellek kategóriájába tartozik. A VAE egyesíti az Autoencoder és a valószínűségi modellezés elemeket. Az Autoencoder egy olyan hálózat, amely képes tömör kódot létrehozni az input adatokból, majd azt kibővíteni a kódból eredeti formájukba. A Variációs Autoencoder további elemekkel kiegészül, amelyek valószínűségi elveken alapulnak.

A VAE két fő részből áll:

Encoder (kódoló): Az encoder feladata egy adott bemeneti adathalmazt (például képeket) átalakítani egy latens térbe tartozó valószínűségi eloszlásba. Azaz, az input adatokat egy valószínűségi térbe képviseli, ahol minden pont (kód) egy különböző lehetséges reprezentációt jelent.

Decoder (dekódoló): A decoder az előző lépésben létrehozott latens kódot visszaalakítja az eredeti adathalmazzá. A decoder tehát a latens térből visszaállítja az eredeti adatokat.

Az egyik kulcsfontosságú jellemzője a VAE-nek az, hogy a latens tér elemei valószínűségi változók. Ez azt jelenti, hogy a VAE nem csak egy konkrét latens reprezentációt generál, hanem egy teljes valószínűségi eloszlást a latens térben. Ezáltal a látens tér tartalmaz egyfajta randomizációt, ami azt biztosítja, hogy ne alakuljon ki az Encoder és a Decoder között szótározás. Ez azért kiemelkedően fontos, mert enélkül kialakulhatna egy olyan rendszer, hogy az Encoder minden bemenetet ugyanarra a látens tér beli reprezentációra kódol és a Decoder, pedig ezt mindig tökéletes visszaállítja az eredeti bemenetté, mert megtanulja, hogy melyik bemenetnek mi a kimeneteli párja. Ez azért okozna hatalmas problémát, mert ha egy a Modellünk álltál nem ismert bemenetet adnánk neki (nincs a tanító adathalmazban), akkor a nem tudna vele mit kezdeni vele és a gyakorlatban használhatatlanná válna. Ezért van szükség arra, hogy a látenstér egy eloszlást fejezzen ki. Ez különbözteti meg az Autoencodert a Variációs Autoencoderektől.



7.ábra Variációs Autóenkóder architektúra
(https://lilianweng.github.io/posts/2018-08-12-vae/)

Ezenkívül még egy kiemelendően fontos tulajdonsága, hogy az Encoder és a Decoder elválasztható egymástól és külön-külön is egy önálló szerepet tudnak betölteni. Ez azért nagyon jó nekünk, mert a látens tért megfigyelhetjük is és mi is tudunk irányítottan saját látens vektorokat létrehozni és ezt tovább adni a Decodernek, ami egy olyan új fajta kimenetet fog tudni nekünk generálni, amit a tanító adathalmaz nem tartalmazott, azonban egy új teljes értékű adat. Tehát képesek lehetünk a Decoder és a látens tért segítségével új adatot generálni, ami egy kiemelkedően, ha nem a legfontosabb tulajdonsága.

5.2 Modellekben felhasznált rétegek

5.2.1 Linear/Fully-Connected Layer

A **Lineáris réteg**, más néven **teljesen összekapcsolt** réteg vagy **sűrű réteg**, az egyik alapvető összetevője a neurális hálózatoknak, különösen a mesterséges neurális hálózatoknak (ANN) és a mélytanulási modelleknek.

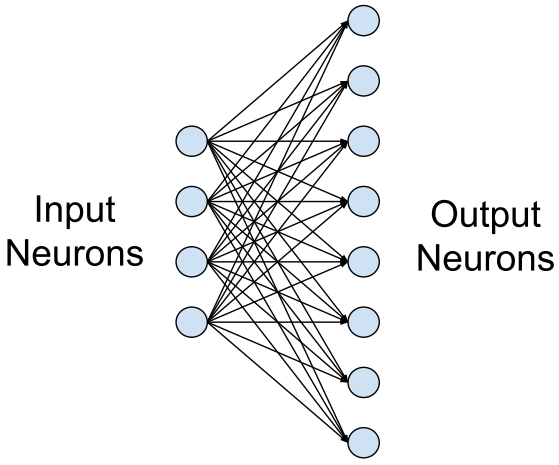
Lineáris transzformáció: Egy lineáris réteg lineáris transzformációt hajt végre a bemenetén. Matematikailag ezt az alábbiak szerint lehet kifejezni:

5. ábra. VAE felépítése

$$y = Wx + b$$

ahol:

- y a kimeneti vektor.
- W egy súlymátrix.
- x a bemeneti vektor.
- b egy eltolási vektor (bias).



8. ábra Lineáris/Teljesen összekapcsolt réteg (https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html)

5.2.1.1 Működés

1. **Súlyok(W):** Egy mátrix, amely tartalmazza a tanulható paramétereket. A bemeneti vektor minden egyes eleme megszorzódik a megfelelő súllyal, és az eredmények összegződnek.
2. **Eltolás (b):** Egy vektor, amely hozzáadódik a bemenetek súlyozott összegéhez, további szabadságfokot biztosítva.
3. **Kimenet:** A súlyozott összeg plusz az eltolás eredménye, amely egy transzformált kimenetet ad.

5.2.2 ReLu Activation Function

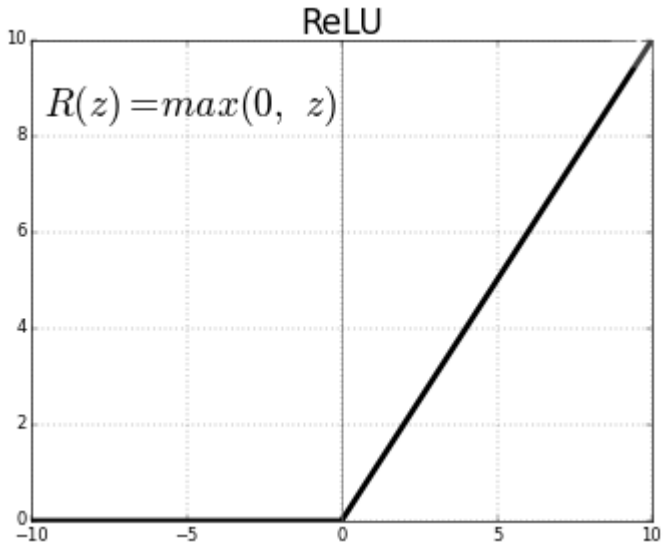
A **ReLU** (Rectified Linear Unit) egy aktivációs függvény, amelyet széles körben használnak mélytanulási modellekben, különösen a konvolúciós neurális hálózatokban (CNN-ek). A ReLU egyszerűsége és hatékonysága miatt népszerű.

A ReLU egy nemlineáris függvény, amely az alábbi módon működik:

$$\text{ReLU}(x) = max(0, x)$$

Ez azt jelenti, hogy:

- Ha a bemenet(x) pozitív vagy nulla, a kimenet(x).
- Ha a bemenet negatív, a kimenet nulla.



9. ábra ReLu aktivációs függvény
(https://www.quora.com/What-is-the-ReLU-layer-in-CNN)

5.2.2.1 Tulajdonságok

1. **Egyszerűség:** A ReLU nagyon egyszerű számítási műveletet végez, ami gyorsabbá teszi a számításokat és hatékonyabbá teszi a tanulási folyamatot.
2. **Nemlinearitás:** Habár a ReLU maga lineáris a pozitív tartományban, a 0 alatt történő levágás (clipping) nemlinearitást ad hozzá, amely lehetővé teszi a neurális hálózat számára, hogy összetettebb mintázatokat tanuljon meg.

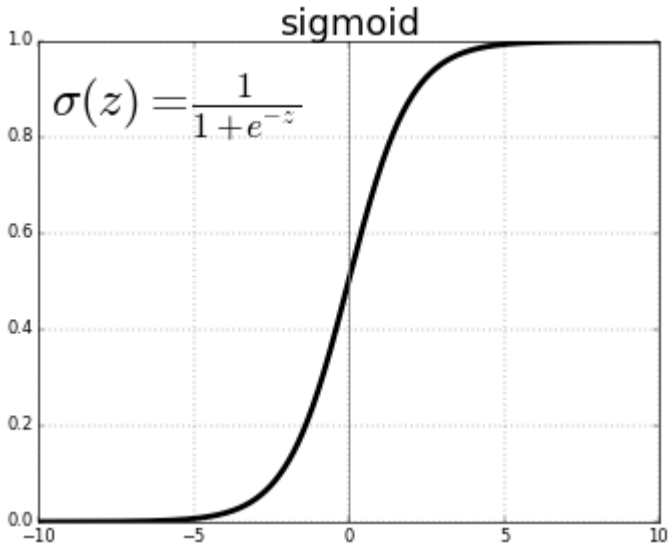
3. **Sparse Aktiváció:** A ReLU hajlamos sparsítást (ritkítást) eredményezni, mert a negatív bemeneteket nullára állítja. Ez növeli a hálózat hatékonyságát és csökkenti a számítási költségeket.

5.2.3 Sigmoid Activation Function

A **szigmoid aktivációs függvény** egy másik fontos aktivációs függvény, amelyet gyakran használnak neurális hálózatokban, különösen a mélytanulás korai szakaszaiban. A szigmoid függvény minden bemenetet egy 0 és 1 közötti értéktartományba transzformál.

A szigmoid függvény matematikai formulája:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



10. ábra Sigmoid aktivációs függvény (https://www.quora.com/What-is-the-ReLU-layer-in-CNN)

5.2.3.1 Tulajdonságok

1. **Kimeneti Tartomány:** A szigmoid függvény kimenete mindig 0 és 1 között van, ami hasznos lehet a valószínűségek modellezésénél.
2. **Simaság:** A szigmoid függvény sima és folytonosan differenciálható, ami elősegíti a gradiens alapú optimalizációt.

5.2.4 Kullback-Leibler divergence

A **Kullback-Leibler divergencia** (KL divergencia vagy KL-divergence) egy mérték, amely két valószínűségi eloszlás közötti különbséget vagy távolságot méri. A KL divergencia a valószínűségelméletben és az információelméletben használatos eszköz, amely megmutatja, hogy mennyire különbözik egy eloszlás egy másik referenciális eloszlástól.

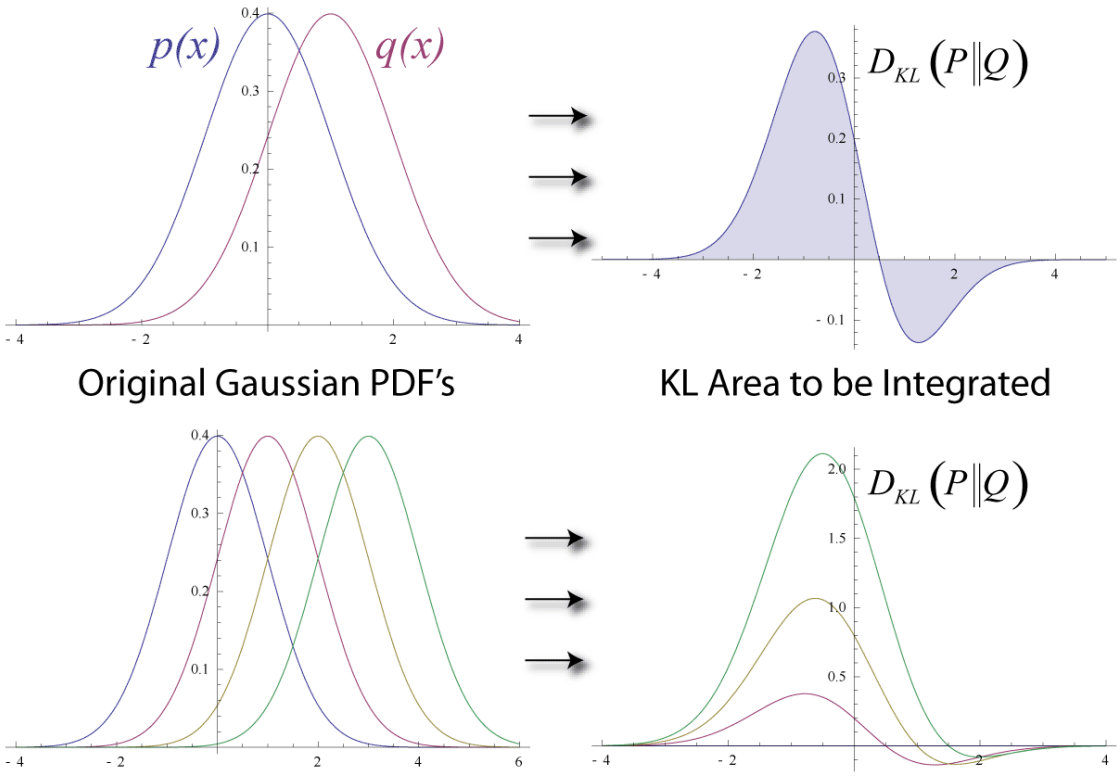
5.2.4.1 Definíció

Legyen P és Q két valószínűségi eloszlás ugyanazon valószínűségi téren. A KL divergencia P eloszlás és Q eloszlás között a következőképpen definiálható:

$$D_{KL}(P \parallel Q) = \sum_x P(x) log(\frac{P(x)}{Q(x)})$$

vagy a folytonos esetben:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} P(x) log(\frac{P(x)}{Q(x)}) dx$$



11. ábra KL-divergencia függvénye (https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence#/media/File:KL-Gauss-Example.png)

ahol:

- P és Q valószínűségi eloszlások (diszkrét esetben) vagy sűrűségfüggvények (folytonos esetben).
- $P(x)$ és $Q(x)$ a valószínűségi eloszlások valószínűségi tömegfüggvényei vagy sűrűségfüggvényei egy adott *x* pontban.
- log a természetes alapú logaritmus.

5.3 Készített modellek

Első nekifutásként úgy döntöttem, hogy magamtól felépítek egy modellt a nulláról, mert ebben még nem volt sok tapasztalatom és VAE-t még sosem készítettem, illetve egy elég speciális problémát kellet megoldanom és nehéz volt egy olyan előre elkészített modellt találni, amit rátudok illeszteni az adatra.

5.3.1 Saját modell

Tapasztalat hiányába kutatást végeztem, hogy érdemes egy VAE-t felépíteni, milyen problémára hasonlít az adatom, milyen rétegekből lenne célszerű felépíteni. Végül az alábbi módon építettem fel a modellemet:

5.3.1.1 Encoder

Az Encoder esetébe 3 Lineáris rétegeket használtam ReLu aktivációval. A végén pedig a VAE-knél megszokott módon egy mean és log-variance paraméterekkel ellátott randomizáció segítségével egy eloszlássá alakítja a látens teret.

5.3.1.2 Decoder

A Decoder esetében ugyanazokat a rétegeket tartalmazza, mint az Encoder azzal a különbséggel, hogy a végén először egy ReLu aktivációs függvényt alkalmaztam, de később miután az adat One-Hot-Encoding-on esett át Sigmoid aktivációs függvényre váltottam.

5.3.1.3 Konklúzió

A modellem nagyon alacsony pontossággal rendelkezett, ezért az adaton is végeztem további változtatásokat, illetve a modellemen is próbáltam javítani, azonban nem találtam pontos okot arra, hogy mi okozza a hibát és nem sikerült érdemileg javítanom a pontosságon, ezért arra a döntésre jutottam, hogy leváltom a modellemet.

5.3.2 MNIST Modell konvolúció nélkül

Az adat augmentációnál eljutottam arra a pontra, hogy egy utat egy 6x70-es vektorral reprezentáltam, ami azért kulcs fontosságú, mert az gyakorlatilag egy kép. Lehet rá képként gondolni, úgy néz ki, mint egy kép és a legfontosabb, hogy fel is lehet dolgozni képként, ami azért nagyon fontos, mert képekkel rengetegen foglalkoznak és tele van példamegoldásokkal az internet. Úgyhogy most már nem voltam teljesen magamra ítélve, nem kellett nulláról felépítenem magamtól egy modellt, hanem keresnem kellett, egy olyan modellt, amivel képeket dolgoznak fel VAE-val és állítanak vissza.

5.3.2.1 MNIST

Az MNIST adathalmaz egy népszerű adatkészlet a gépi tanulásban és a képfeldolgozásban, amely kézírásos számjegyeket tartalmaz. 60000 tanító adatot és 10000 teszt adatot tartalmaz. Az összes adat 28x28 pixeles képekből áll.

Adathalmazt egyszerűbb keresni, mint közvetlen modellt, mert az számít, hogy mi a probléma, amit meg akarsz oldani, ami jelen esetünkben képeket feldolgozni VAE-val és tudtam, hogy az MNIST egy nagyon népszerű adathalmaz az ilyen problémákhoz.

5.3.2.2 Választott modell

Nagyon könnyű volt modellt találni a problémához, azonban nem volt megfelelő bármilyen, mert a képfeldolgozás esetén tipikusan konvolúciót alkalmazunk, ami az adott pixelek szomszédsági viszonyait emeli ki, mintha ráközelítenénk egy kép adott részére. Ez viszont nekem nem pozitív tulajdonság, hanem nagyon negatív, mert az én utaim esetében, a koordináták közötti közölség semmilyen információt nem hordoz nem is beszélve arról, hogy az adat nagyrésze 0. Ezért nagyon nem lenne jó, ha modell erre rá akarna tanulni.

Végül sikerült találnom a konzulensem segítségével egy olyan modellt [8], ami nem alkalmaz konvolúciót és megoldja az MNIST-es problémát VAE-val, úgyhogy erre esett a választás.

5.3.2.3 Encoder

Két teljesen kapcsolt rétegből áll, amik LeakyReLu aktivációs függvénnyel vannak összekapcsolva.

A LeakyReLu annyiban tér el a fentebb említett ReLu-tól, hogy a negatív érékeket nem nullára állítja, hanem lineárisan egy ahhoz közeli értékhez.

5.3.2.4 Latent Mean és Variance Layer:

TODO?

Látens tér egy 2 dimenziós vektor.

5.3.2.5 Reparametrizációs Függvény:

TODO?

5.3.2.6 Decoder

Három teljesen kapcsolt réget, amelyek LeakyRelu aktivációs függvénnyel vannak összekapcsolva, az utolsó rétegben pedig a Sigmoid aktivációs függvény található.

5.3.2.7 Konklúzió

Az itteni modell nem sokban tér el az általam készített modelltől, ami arra következtet, hogy valamilyen hibát követhettem el az implementálása során, mert a két modell közti eredmény beli különbség túl nagy.

6 Eredmények

Az modell kimenetét rendkívül sokféleképpen lehet mérni és rengeteg különböző statisztikát lehet rá készíteni. Ezeket most két nagyobb csoportba soroltam és azonfelül még pár kisebbre.

6.1 Útvonal visszaállítása

Legelsőnek azt teszteltem, hogy ha adunk egy utat a modellnek, akkor azt milyen pontossággal tudja visszaállítani. Ebben is két külön kategória van:

1. **Kereszteződés**: Minden egyes utcára adott predikciót összehasonlítása az eredeti adatban szereplő utcával.
2. **Útvonal**: Milyen pontossággal állította vissza a modell a teljes útvonalat.

Kereszteződés pontossága: 78%

Útvonal pontossága: 31%

A fontosabb cél nem az utak visszaállítása volt, hanem új utak létrehozása, aminek a mérésére két új mérőszámot vezetek be.

6.1.1 Novelti

Azt mondja meg, hogy a visszaállított út milyen %-ban szerepel az eredeti adathalmazban. Ezt a **Kereszteződésre** és az **Útvonalra** és kiszámoljuk.

TODO, nem lett kiszámolva

6.1.2 Validiti

Azt mondja meg, hogy a visszaállított út milyen %-ban szerepel a térképen, aminek az érték készlete nagyobb egyenlő az adathalmazzal. TODO matekosan Ezt a **Kereszteződésre** és az **Útvonalra** és kiszámoljuk.

Kereszteződés pontossága: 90%

Útvonal pontossága: 69%

6.2 Látens tér mintavételezése

A látenstér mintavételezése számunkra a legfontosabb, mert ezzel tudunk egy biztos pontosságot mérni a modell működéséről és teljesen új útvonalakat generálni. Az egy kiemelkedően fontos tulajdonsága, hogy tisztán a látenstérből generált utak esetén mért pontosságban semmilyen módon nem csalhattunk, véletlen se keverhettük bele a tanító adatot a predikcióba, mert teljesen random a háló bemenete.

Itt kizárólag Noveltit meg Validitit van értelme lemérni, mert nincs mihez hasonlítani az eredményt nem volt semmiből generálva. Kétféle módon generáltam a látenstérből utakat:

1. **Teljesen véletlenszerű**: A látenstért generáló függvényt mean = 0, logvariancia =1 hívom meg, ami egy teljesen random vektort ad vissza a látens térből.
2. **Útvonal környezete**: Egy random útvonalból generált látens vektort egy állítható **r** paraméterrel randomizálom.

6.2.1 Novelti

TODO

6.2.2 Validiti

Teljesen véletlenszerű:

- Kereszteződés pontossága: 88%
- **Útvonal pontossága: 61%**

Útvonal környezete:

- Kereszteződés pontossága: 84%
- **Útvonal pontossága: 48%**

7 Összefoglalás

Az eredmények kifejezetten elégedett vagyok, de még további méréseket kell végezni, hogy pontosan mit is jelentenek ezek az eredmények és a más módszereknél jobban vagy jobban teljesítek. Mindenképp fontos megnézni, hogy mennyire skálázható a megoldás, mert ha sokkal több utat szeretnénk hosszabban generálni, akkor nagyon gyorsan megugrik a számolási igény a Mátrixok méretei miatt.

8 Irodalomjegyzék

[1] https://medium.com/@rekalantar/variational-auto-encoder-vae-pytorch-tutorial-dce2d2fe0f5f

[2] https://www.quora.com/What-is-the-ReLU-layer-in-CNN

[3] https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence#/media/File:KL-Gauss-Example.png

[4] http://project.mit.bme.hu/mi_almanach/books/neuralis/ch01s02

[5] https://docs.nvidia.com/deeplearning/performance/dl-performance-fully-connected/index.html

[6] https://en.wikipedia.org/wiki/MNIST_database

[7] https://lilianweng.github.io/posts/2018-08-12-vae/